# System Engineering Challenges of Real-Time Simulation for Mars Smartlander Entry, Descent and Landing

Bryan J. Martin*
*Jet Propulsion Laboratory, Pasadena, CA*
David. A. Henriquez†
J. Bob Balaram‡
Garett. A Sohl†
and
Marc. I. Pomerantz§

DSENDS is an in-development spacecraft simulator for Entry, Descent, and Landing, developed initially through DNP funding at JPL and currently slated for use in the Mars Smart Lander mission as an analyis and verification tool. Based on the JPL-developed **Dshell** simulator, the need to consolidate and combine tools from disparate sources and with varying degrees of completeness has presented unique challenges. This paper discusses some of those challenges and and their solutions.

## INTRODUCTION

The Jet Propulsion Laboratory (JPL) is developing a high-fidelity, real-time spacecraft simulator for Entry, Descent and Landing (EDL) on planetary and small bodies. This simulator, DSENDS (Dynamics Simulator for Entry, Descent and Surface landing), is an EDL-specific extension of the JPL Darts/Dshell multi-mission spacecraft dynamics and devices simulation toolkit which is in use by missions such as Cassini, Galileo, SIM, and Starlight. The broad scope of possible missions for this tool requires that we maintain a high level of generality and flexibility in our system design. The first major test of the DSENDS simulator is in support of the Mars SmartLander mission, which will use DSENDS in a real-time environment (**Mars-DSENDS**).

The DSENDS simulator interfaces with three general classes of tools. *COTS* tools such as StethoScope, daVinci, Graphviz, and Matlabs' Simulink require use of a stable and standardized API. *JPL internal* tools such as the visualization programs Dview and Dspace, are integrated with the Dshell environment yet run as external processes using a coordinated, semi-asynchronous form of IPC. Finally, *instrument, system component, and environmental models* developed by disparate sources within and without JPL each present challenges due to the need to develop a unique interface or wrapper that takes into account that they are typically mission components that are under development.

Within this paper we discuss the integration and system engineering issues for a selection of the above elements into the DSENDS simulator.

## SYSTEM ENGINEERING

Handling multiple sim configurations

- Modularization of s/c models
- Landings sites
- Planet data

## SPACECRAFT DYNAMICS

The multi-mission capabilities of Darts/Dshell allow easy reconfiguration of DSENDS to simulate a wide range of spacecraft configurations. Mass properties, thruster sizes and thruster locations can be changed to reflect different design concepts. DSENDS simulations

*Senior Staff Engineer, Engineering and Science Directorate, Avionic Systems and Technology Division, Autonomy and Control Section, Simulation and Verification Group.

†Staff Engineer, Engineering and Science Directorate, Avionic Systems and Technology Division, Autonomy and Control Section, Simulation and Verification Group.

‡SomeOtherDesignation, Engineering and Science Directorate, Avionic Systems and Technology Division, Mobility Systems Concept Development Section, SomeOtherGroup

§Member of Technical Staff, ARCO Service Corporation

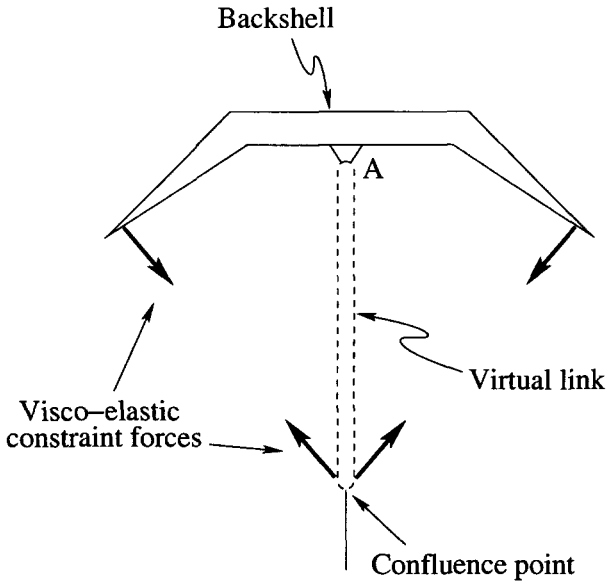


**Fig. 1 Confluence point dynamics**

have been created based on an early '07 prototype design and the current SmartLander reference design. Even though the current reference design is in flux, DSENDS provides the flexibility needed to evolve with any changes.

### Dynamic Spacecraft Reconfiguration

One significant challenge in EDL simulation is managing the transitions between the multiple phases of EDL, which often require a structural change in the simulation model to reflect new spacecraft configurations. Heat shield separation and parachute deployment are examples of these transitions. These changes are conceptually simple, but require automated methods to maintain the continuity and accuracy of the end-to-end simulation. Using a state machine (see section ), DSENDS allows dynamic reconfiguration of the spacecraft model and correctly propagates the old system state to the new model. This ability to reconfigure the simulation model allows DSENDS to provide varying levels of fidelity during each phase. Integration step size can also be varied during the simulation, allowing the user to choose particular phases to simulate at high fidelity while maintaining continuity with the rest of the simulation.

### Tether Dynamics

After parachute deployment, the flight train system includes several flexible tethers. Our tests of simulation methods for these tether lines are based on a Pathfinder-like system. This system includes a backshell-lander system involving three lines and a single confluence point (see Figure 1). A similar system is used for the backshell-parachute tether line system.

A visco-elastic model for the individual tether line is used. The constraint forces imposed by the individual tethers dictate the motion of the confluence point. The high frequency dynamics of the tether lines requires a small integrator step size to insure stability. Although variable step integrators are more accurate, they do not provide any guarantee of timing. This can prove unacceptable in a real-time environment.

Our early testing of visco-elastic tether systems showed similar qualitative results for both fixed and variable step integrators. These qualitative behaviors may be adequate for general simulation. Shown in Figure 2 are some simulation results for a triple bridle system.

Simulation speed of the variable step techniques is highly dependent upon the state of the tethers (see Figure 3).

Variable step integrators reduce the step size when a tether transitions between slack and taut (between t = 1 and t = 5 in test plot). This provides increased accuracy at the cost of simulation speed, making the variable step integrators slower than a fixed step near these transitions. However, as the system settles and the tether lines remain taut, a variable step integrator can increase the step size and achieve faster performance compared to a fixed step method (t ¿ 6 in test plot).

The spring-damper tether models have been incorporated into the DSENDS simulation. DSENDS currently uses a fixed step integrator and requires a reduction in step size when using the high fidelity tether models.

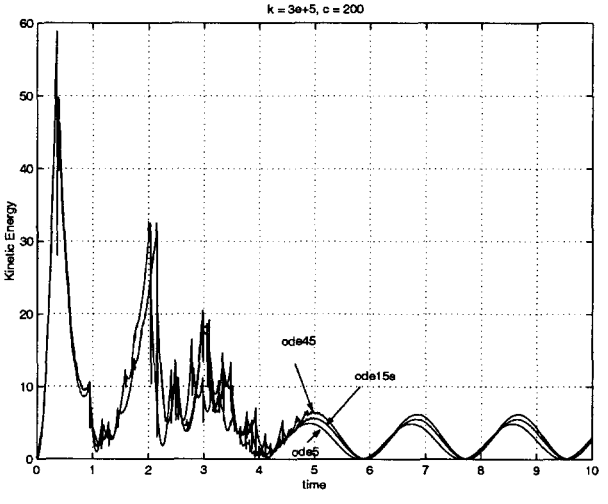


**Fig. 2 Tracking results for different integration schemes.**

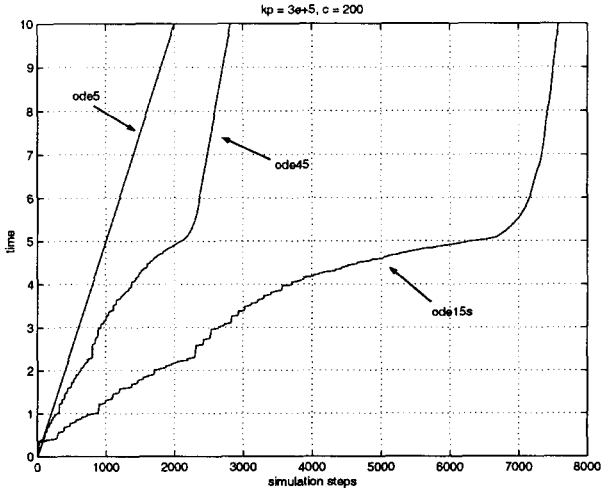


**Fig. 3 Integration effort for different integration schemes.**

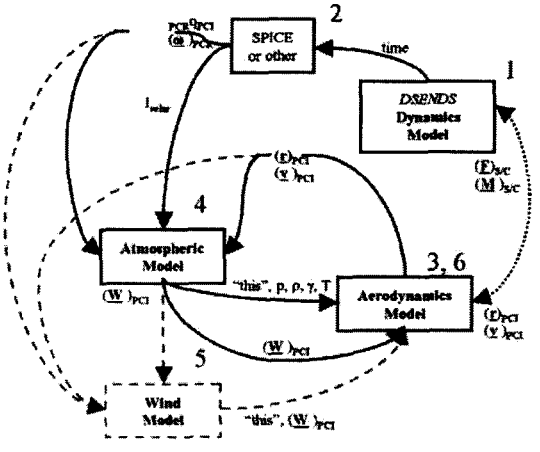


**Fig. 4 Connectivity between aerodynamic models.**

## AERODYNAMIC MODELING

Aerodynamic forces and moments are fundamental for an Entry, Descent and Landing simulator. During, EDL, an aerodynamic system will experience aerodynamic forces as it traverses from the thin outer atmosphere (governed by free molecular flows) to the atmosphere at the landing site (governed by continuum fluid flows). Also, the aerodynamic system decelerates from hypersonic velocities (in which molecular dissociation dominates the boundary layer) to subsonic velocities on the order of minutes. Thus, the challenge for the DSENDS EDL simulator is how to accommodate such disparate flow and velocity regimes in a real-time environment.

Another challenge has been how to leverage existing aerodynamic models from legacy codes used for trajectory simulations (e.g. POST). These legacy codes are well suited for Monte Carlo mission analyses but were never intended to operate in a real-time simulation environment. Although these codes tend to be monolithic, it is possible to isolate the functions in these validated and trusted codes that are used to compute aerodynamic forces and moments. These functions could then be imported into the DSENDS simulator, so long as they do not violate the real-time requirement and can be hosted as a model in the DSENDS aerodynamic modeling architecture.

Consequently, DSENDS had to architect the manner in which it models aerodynamics so that it could 1) operate in a real-time environment and 2) inherit legacy or validated aerodynamics models. Real-time considerations drove the aerodynamics modeling architecture to restrict the computational overhead of the models and to allow for scalable fidelity. In order to meet these requirements, DSENDS aerodynamics modeling architecture decomposed the aerodynamics into three separate models: 1) Aerodynamics Model (encapsulates the aerodynamic coefficients for the given aerodynamic system), 2) Atmospheric Model (encapsulates the ambient atmospheric conditions) and 3) Wind Model (any local wind velocity not captured in the Atmospheric Model. Common interfaces and functional requirements were made for these three groups of models so that they would correctly interoperate. The fidelity and computational overhead of an aerodynamic model would then depend on which set of models were instantiated for a given DSENDS EDL simulation. Also, different kinds of legacy codes could be encapsulated one of these three types of models.

Currently, DSENDS has the following catalog of models.

- Aerodynamics Models
  - Linearized, Axisymmetric Aerodynamics
  - Interpolated, Axisymmetric Aerodynamics
  - Viking-based, Hypersonic Aerodynamics developed at LaRC (includes Mars GRAM 3.37)
- Atmospheric Models
  - Fitted Reference Martian Atmosphere
  - Mars GRAM
- Wind Models
  - Under Development

The Viking-based hypersonic model listed above was intended for use in extremely high-fidelity atmospheric entry simulations, and was not optimized for use in a real-time environment, although we expected it to be

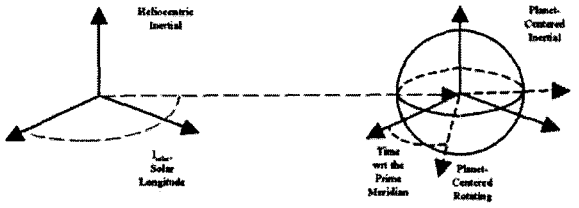

**Fig. 5 Celestial body frames.**

efficiently designed and implemented. Our integration and testing efforts therefore concentrated on efficient use of the library as well as validation of our results.

Our preliminary tests incorporate the LaRC hypersonic model into **Mars-DSENDS** using and axis-symmetric body model. The flight path and time-dependent dynamic quantities (acceleration, dynamic pressure, etc) were compared against the pre-existing low fidelity models previously used in DSENDS . These tests have been performed on hardware running a non-real-time OS (Solaris), and so have been used only to measure theoretical or relative performance. The low-fidelity model showed qualitatively similar results but did not have the frequency content of the LaRC model.

The following sections discuss the requirements and implementation of the DSENDS aerodynamics modeling architecture.

## System Engineering Issues Related to Aerodynamics Modeling

### Requirements for Atmospheric Models

Atmospheric models are either simple mathematic model or a database of atmospheric properties (e.g. density, pressure, temperature, wind velocity, etc.) that vary with time. Spatial-temporal variations in the local atmospheric properties are due to the uneven, radiative heating of the planet and influence of the planetary terrain and consequently vary with local time (i.e. planetary attitude), solar longitude (i.e. season) and with planetary latitude and longitude.

Although terrain information may be convolved into an atmospheric model, no atmospheric model shall be expected to produce terrain elevation. Synchronization between atmosphere and terrain models shall be done external to the atmospheric model and is expected to be solely through relative position of the aerodynamic system (i.e. latitude, longitude and height) with respect to a planet-centered, rotating frame (i.e. PCR).

Since atmospheric models are expected to be relative to the planetary surface, i.e. vary with latitude, longitude and height above the surface, each atmospheric model must have access to a function or object to convert its planet-centered, rotating frame (PCR) data into a planet-centered, inertial frame (PCI), which makes the data more compatible with the rest of the DSENDS EDL simulation.

As mentioned above, atmospheric models are expected to have data that varies with solar longitude and local time (i.e. planetary attitude and rate) or some extraterrestrial equivalent of GMT. However, atmospheric models shall not be expected to maintain such planetary attitude and rate. Therefore, planetary attitude and rate must come from some external object that maintains such states (e.g. a planetary attitude dynamics object or SPICE object).

Atmospheric models may produce wind velocities as a function of the relative position of the aerodynamic system above the planetary surface. As mentioned earlier, the atmospheric model shall have access to a function or object to transform the wind velocities into a standard PCI frame. The atmospheric model must also correctly add the planetary rotation to wind velocity so that the inertial wind velocity represents the total kinetic energy of the flow field encountered by the aerodynamic system.

Although there are classes of atmospheric models that can be modeled by a set of ODE, such models will not be used in DSENDS . DSENDS has a requirement to be a real-time simulator and using ODE-based atmospheric models can easily violate that requirement. Therefore, ODE-based atmospheric models cannot be used at runtime within DSENDS . However, the spatial-temporal data produced by ODE-based atmospheric models could be an input to some atmospheric model that interpolates its spatial-temporal data. In other words, DSENDS atmospheric models shall only have a functional dependence on its input, regardless of whether the underlying atmospheric model uses interpolated data or an algebraic equation.

### Requirements for Wind Models

Simple wind models are those that model the diurnal and semi-diurnal cycles in the wind (due to diurnal solar radiative heating) as the sum of sinusoidal functions in time. High fidelity wind models in the atmospheric sciences use a set of spatial-temporal partial differential equations that are entered into an estimation process (i.e. Kalman Filter) where atmospheric and terrain measurements are used to refine the results so that a realistic wind field is produced with spatial-temporal variations. Many of these effects are available in certain atmospheric models (i.e. Mars GRAM) with varying degrees of fidelity. It is for this reason that DSENDS wind models shall only

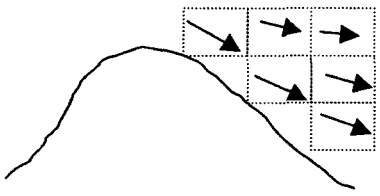


**Fig. 6 Semi-diurnal flow near a mountain.**

compute local perturbing wind velocities for a specified ground-fixed atmospheric volume (i.e. latitude, longitude and height ranges). This requirement facilitates correlation between a wind model and a high fidelity atmospheric model without the same wind effects being doubly applied or erroneously cancelled. Furthermore, DSENDS wind models shall maintain parameters that define the PCR-fixed region of influence to which the wind model is valid. The wind model shall also provide a public function or object to return these parameters so that the perturbed wind velocity can be correlated with the planetary terrain for the same PCR-fixed volume. These parameters can either be hard-coded into the model or set by the user. The wind model shall have access to a function or object that converts its PCR velocity vectors into the PCI frame, so that its output is compatible with the DSENDS environment. Since only perturbations are computed in DSENDS wind models, a wind model shall not be required for running a DSENDS EDL simulation. Spatial-temporal wind models have a set of ODE's that must be solve for the entire wind field throughout its PCR extent at various grid points. The perturbed local wind velocity is then queried for a specific location. If the specified location was between the model's grid points, the model interpolates the wind velocity from neighboring grid points. Such a model could be a large computational burden for DSENDS and may violate the real-time requirement for the DSENDS simulator. However, since such models are desired for modeling localized wind effects near mountains, craters or valleys, DSENDS wind models shall have a bounded region of influence such that its computation overhead does not violate the DSENDS real-time constraints. In addition, DSENDS wind models shall have access to functions or objects that allow it to maintain continuous states and functions or objects that allow it interpolate between grid point solutions.

The subsequent discussion will focus on the requirements for implementing a simple spatial wind model, which should not preclude the implementation of more complex wind models. A perturbing wind model can be developed using a PSD that only varies with altitude and does not require solving a set of ODE's. After the coefficients to the discrete Fourier series are computed, the wind model could take a PCI position, convert it into an altitude and compute the perturbing wind at that altitude. Additionally, truncating the Fourier series easily bounds the computational overhead of this wind model. Many of the variable used to compute the Fourier series (e.g. the fundamental frequency, number of harmonics, etc.) can be model parameters, thereby making such a wind model well suited for generic application. Consequently, DSENDS wind models shall use model parameters to increase the model's potential applicability in DSENDS EDL simulations.

### Requirements for Aerodynamics Models

Aerodynamic are modeled as a set of dimensionless coefficients that are relative to a body-fixed reference frame on an aerodynamic system. The reference frame defines a coordinate system in which forces and moments are applied. This reference frame is very likely to be different from the mechanical reference frame. Therefore, aerodynamics models shall maintain a transform that allows aerodynamic forces to be mapped from the aerodynamic reference frame into the mechanical reference frame of the aerodynamic system.

The aerodynamic reference frame shall have the +x direction is along the axis of symmetry and is nominally antiparallel to the freestream velocity when the total angle of attack is zero (i.e. +x points into the "wind"). The +y direction shall be defined as the "pitching axis" and shall point such that a positive rotation (i.e. counterclockwise) about +y shall produce a positive angle of attack. +z shall be the cross product of +x and +y. This definition of the aerodynamic reference frame is general and can be applied to any aerodynamic system.

So far, the discussion on modeling requirements for DSENDS aerodynamics models has been general. The following discussion will focus on the requirements needed for modeling axisymmetric aerodynamic systems. Axisymmetric aerodynamics models are a class of aerodynamics models that are not sensitive to an angular displacement and angular rate about the axis of symmetry. A seemingly limiting choice of aerodynamic systems, axisymmetric aerodynamics models can be used to model most EDL aerodynamic subsystems (e.g. entry vehicle, parachute, etc.). Therefore,

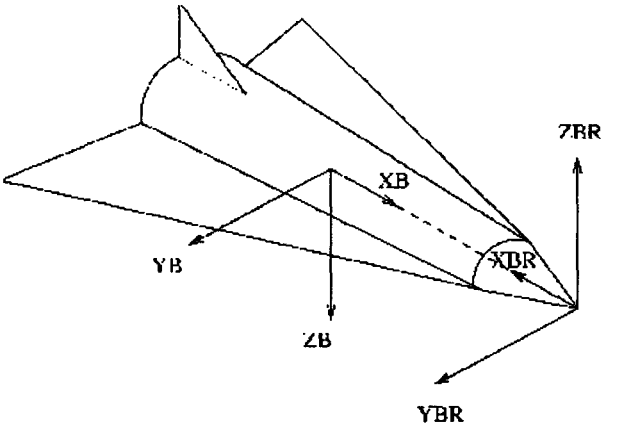


**Fig. 7 Aerodynamic body reference frames**

DSENDS development focused on developing these models, but the interfaces and design of the DSENDS atmospheric and wind models will not preclude future lifting body aerodynamics models.

Although aerodynamic systems experience positive and negative total angles of attack, most aerodynamic coefficients databases for axisymmetric vehicles tend to only list coefficients for positive angle of attack. This convention assumes that the aerodynamic forces and moments are applied in a frame such that the total angle of attack is positive. As a result, axisymmetric aerodynamics model shall determine its aerodynamic reference frame at runtime. The frame shall maintain +x along the axis of symmetry but +y shall be chosen such that the total angle of attack the angle is positive; +y and +z directions will vary as the incident flow field. The transform maintained by the axisymmetric aerodynamics model shall be used to define the +x with respect to the mechanical reference frame.

Lastly, the axisymmetric aerodynamics model shall have access to functions or objects that allow it to ascertain the 6 DOF inertial state of the aerodynamic system. Furthermore, the axisymmetric aerodynamics model shall have access to functions or objects that allow it convert its 6 DOF state knowledge into total angle of attack within the aerodynamics reference frame.

## Software Engineering for Atmospheric Models

### Interfaces

In the following interface description, assume that the base data type for variables and arrays is a double precision floating point type (i.e. double) unless otherwise specified.

The interfaces for DSENDS atmospheric models, as derived from its requirements, must have the following inputs and outputs. Atmospheric models shall have as its input: 1) a three element array for the PCI position

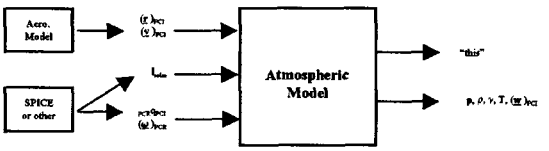


**Fig. 8 Atmospheric model interfaces**

of the aerodynamic system, 2) a three element array for the PCI velocity vector of the aerodynamic system, 3) solar longitude for the planet, 5) a four element array for the quaternion that defines the transformation from the PCI frame to the PCR frame and 6) a three element array for the rotation rate of the PCR frame with respect to the PCI frame. And the output of a DSENDS atmospheric model shall be: 1) an unsigned integer that represents the address of the atmospheric model (i.e. a "this" pointer), 2) a local density, 3) a local pressure, 4) a local temperature, 5) the local specific heat ratio (i.e. ?) and 6) a three element array for the local wind velocity in the PCI frame. The justification for a "this" pointer output is made in the following section.

### Implementation

DSENDS defines a C++ model class called flow model that only has a functional dependence on its input and has no continuous states (i.e. cannot define a derivative). The flow model class was chosen as the base class for DSENDS atmospheric models in order implement atmospheric models that have a functional dependence on its input.

The DSENDS atmospheric models will communicate with the DSENDS aerodynamics models via inherited class types called flowIns and flowOuts. These types are defined in the DSENDS model base class, from which the DSENDS flow model class is derived, and allow DSENDS models to read and write data to signal buffers. Two DSENDS models can communicate with each other if they are connected to the same set of signal buffers. Therefore, the DSENDS atmospheric model was implemented to read the PCI position and velocity vectors, the quaternion from PCI to PCR and the angular rate of the PCR frame from the following similarly named flowIns object:

1. flowIns.positionPCI[3]
2. flowIns.velocityPCI[3]
3. flowIns.quatPCItoPCR[4]
4. flowIns.ratesPCR[3]

And the atmospheric model was implemented to write its address, the local density, the local temperature, the local pressure, the local specific heat ratio and the local wind velocity in the PCI frame to the following similarly named flowOuts object:

1. flowOuts.atmObject
2. flowOuts.atmDensity
3. flowOuts.atmTemp
4. flowOuts.atmPressure
5. flowOuts.atmHeatRatio
6. flowOuts.atmWindPCI[3]

The DSENDS atmospheric model outputs its "this" pointer in order to associate it with a DSENDS aerodynamics model. The aerodynamics model is the forcing function on an aerodynamic system and in order for it to compute the proper forces and moments to apply, it must know the local atmospheric properties for the current inertial state of the aerodynamic system. As will be further elucidated below, the DSENDS atmospheric model will execute by a direct command from the DSENDS aerodynamics model (i.e. via a public member function of the atmospheric model class). Upon receiving the execution command, it will read its input from the flowIns object and write the local atmospheric properties to its flowOuts object. Hence, the aerodynamics model shall always have the correct local atmospheric properties, so long as the aerodynamics model and atmospheric model are connect to the same set of signal buffers and the aerodynamics model writes its PCI position and velocity prior to executing the atmospheric model.

## Software Engineering for Wind Models

### Interfaces

In the following interface description, assume that the base data type for variables and arrays is a double precision floating point type (i.e. double) unless otherwise specified.

As derived from its requirements, DSENDS wind models shall have the following interfaces for its inputs and outputs. The inputs of a wind model shall be: 1) a three element array for the PCI position of the aerodynamic system, 2) a three element array for the PCI velocity vector of the aerodynamic system, 3) a four element array for the quaternion that defines the transformation from the PCI frame to the PCR frame, 4) a three element array for the rotation rate of the PCR frame with respect to the PCI frame and 5) a three element array for the local wind velocity in the

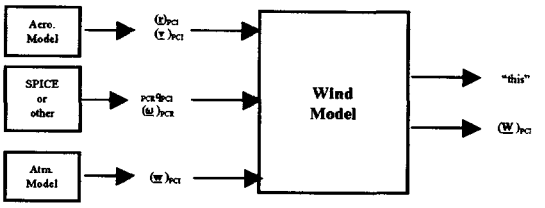


**Fig. 9 Wind model interfaces**

PCI frame (i.e. global atmospheric circulation). The wind model outputs shall be: 1) an unsigned integer that represents the address of the wind model (i.e. a "this" pointer) and 2) a three element array for the perturbed local wind velocity in the PCI frame. The justification for a "this" pointer output is made in the following section.

### Implementation

Unlike the atmospheric model, the DSENDS wind models were implemented using the DSENDS base class that has class types that allow the model to maintain continuous states. The DSENDS sensor model base class was selected because it was assumed that the variations in the local perturbing wind velocity varies in time scales that allow it to be loosely coupled with the aerodynamic forces and moments (i.e. minutes or hours). Consequently, there is no need to solve simultaneously solve a wind model ODE and the equations of motion of the aerodynamic system.

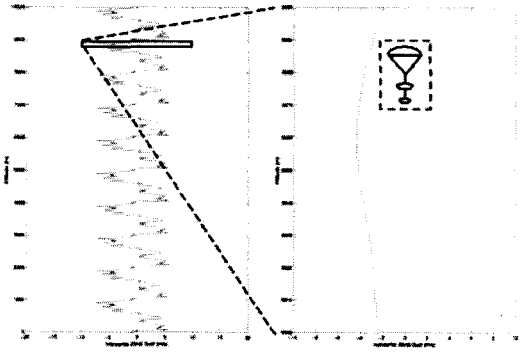

**Fig. 10 Wind model interfaces**

The DSENDS wind models will communicate with the DSENDS aerodynamics models via inherited class types called flowIns and flowOuts (see Software Engineering for Atmospheric Models section for a description of these types and their use). The DSENDS wind model was implemented to read the PCI position and velocity vectors, the quaternion from PCI to

PCR, the angular rate of the PCR frame and the atmospheric wind velocity from the following similarly named flowIns object:

- flowIns.positionPCI[3]
- flowIns.velocityPCI[3]
- flowIns.quatPCItoPCR[4]
- flowIns.ratesPCR[3]
- flowIns.atmWindPCI[3]

And the wind model was implemented to write its address, the local perturbed wind velocity in the PCI frame to the following similarly named flowOuts object:

- flowOuts.windObject
- flowOuts.perturbWindPCI[3]

The DSENDS wind model outputs its "this" pointer in order to associate it with a DSENDS aerodynamics model. The aerodynamics model is the forcing function on an aerodynamic system and it must have the local wind velocity vector for the current inertial state of the aerodynamic system in order to correctly compute the freestream velocity vector and therefore the correct angle of attack. If a DSENDS wind model is connected to a DSENDS aerodynamics model, the DSENDS aerodynamics model will execute the wind model via a public member function of the wind model class. Upon receiving the execution command, the wind model will read its input from the flowIns object and write the local perturbed wind velocity to its flowOuts object. The aerodynamics model must execute the atmospheric model prior to executing the wind model. If no wind model is connected (i.e. NULL address or address equals the atmospheric model address) the aerodynamics model will skip execution of the wind model and will simply read the wind velocity on its flowIns object.

## Software Engineering for Aerodynamic Models

### Interfaces

In the following interface description, assume that the base data type for variables and arrays is a double precision floating point type (i.e. double) unless otherwise specified.

As required, DSENDS aerodynamics models shall have the following input and output interfaces. The inputs of a aerodynamics model shall be: 1) an unsigned integer that represents the address of the atmospheric model (i.e. a "this" pointer), 2) a local density, 3) a local pressure, 4) a local temperature, 5) an unsigned integer that represents the address of the wind model (i.e. a "this" pointer), 6) a three element array for the local wind velocity in the PCI frame and 7) the 6 DOF inertial state of the aerodynamic system. The aerodynamics model outputs shall be: 1) a three element array for the PCI position of the aerodynamic system, 2) a three element array for the PCI velocity vector of the aerodynamic system and 3) the aerodynamic force and moment applied in the mechanical reference frame. The justification for the use of "this" pointers output is made in the following section.

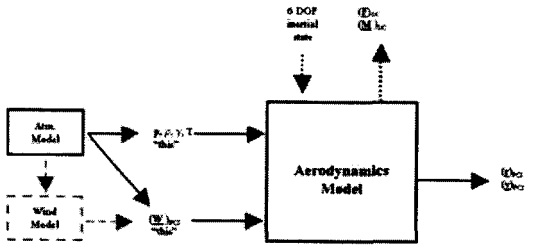


**Fig. 11 Aerodynamic model interfaces**

### Implementation

The DSENDS aerodynamics models were implemented using the DSENDS actuator model base class, which allows the model to apply forces and moments on the DSENDS dynamics model.

The DSENDS aerodynamics models will communicate with the DSENDS atmospheric models and DSENDS wind models via inherited class types called flowIns and flowOuts (see Software Engineering for Atmospheric Models section for a description of these types and their use). The DSENDS aerodynamics model was implemented to read the address of the atmospheric model, the address of the wind model, the local density, the local temperature, the local pressure, the local specific heat ratio and the local wind velocity in the PCI frame from the following similarly named flowIns object:

- flowIns.atmObject
- flowIns.windObject
- flowIns.atmDensity
- flowIns.atmTemp
- flowIns.atmPressure
- flowIns.atmHeatRatio
- flowIns.localWindPCI[3]

And the aerodynamics model was implemented to write its PCI position and velocity vectors to the following similarly named flowOuts object:

- flowOuts.positionPCI[3]
- flowOuts.velocityPCI[3]

The DSENDS aerodynamics model uses the addresses of the DSENDS atmospheric model and the address of the DSENDS wind model in order to associate and communicate with them. After the aerodynamics model gets the 6 DOF inertial state of the aerodynamic system from the DSENDS dynamics model via a function call, it extracts its PCI position and velocity and writes it to its flowOut type, thereby providing inputs to the atmospheric and wind models. The aerodynamics model then proceeds to check its flowIns type for valid atmospheric and wind model addresses. If the address of the DSENDS wind model is null, the aerodynamics model skips execution of the wind model. However, if the address of the atmospheric model is null, the aerodynamic model will print a critical error message and force the DSENDS simulator to exit.

After verifying that it is connected to at least a valid atmospheric model, the aerodynamics model executes the atmospheric model and then the wind model (if valid) using public member functions of the models. The aerodynamics model then reads the ambient atmospheric conditions and wind velocity from flowIns type. The aerodynamics model then takes its inertial velocity and wind velocity to compute the freestream velocity and angles of attack. And then it uses the ambient atmospheric conditions and freestream velocity to compute the Mach number of the aerodynamic system. If the aerodynamics model uses an aerodynamic coefficient database, then the aerodynamics model must use some interpolation functions to determine the aerodynamic coefficients for the given aerodynamic system state. If the aerodynamic model uses some parametric aerodynamic coefficient model, the aerodynamic model then queries the coefficient model by some function call to get the salient aerodynamic coefficients. However the aerodynamics model accesses its aerodynamic coefficients, it takes the coefficients for the given angles of attack and Mach number and then computes the aerodynamic forces and moments to apply to the aerodynamic system.

## TERRAIN MODELING

Terrain models are used by DSENDS for a variety of purposes. These include monitoring the height of the spacecraft over the terrain, determining instrument field-of-views, generating sensor responses for Imaging, Lidar and Phased-Array Radar terrain sensing devices, evaluating the spacecraft kinematic and dynamic response during touchdown, and overall visualization of the simulation. The extent and resolution of the terrain needed by DSENDS varies during the descent profile. At entry, a large swath of terrain is required at coarse resolution. As the spacecraft descends narrower extents of terrain at higher resolution is required. Furthermore, the specific locations at which the terrain is required also varies as a result of uncertainty in the spacecraft trajectory and pointing. For example, during hypersonic entry at Mars an unguided spacecraft has an a trajectory uncertainty of typicall over a 100 km. Similarly, during descent on a parachute, the pointing of the spacecraft can vary by many 10's of degrees as the parachute interacts with the descent induced and atmospheric wind. A typical scenario of terrain usage is shown in Figure 1.

The specific terrain models supported in DSENDS are Digital Elevation Maps (DEMS). This is essentially 2.5 D data and suffices for most EDL simulation needs. The sources of the DEM data are varied. At one level is planet-wide data that may have been obtained by remote sensing from orbit. Such topographic data is available, for example, for Mars using the MOLA data set or for Venus using the Magellan data set. At another level is purely synthetic data. Such data, using fractal generation of landscape features such as craters and rocks overlayed on a base topography, is useful for performing system tests where real data is not available. Such synthetic data is available from a variety of sources[12].[3] Finally, we have data that is intermediate between the two, where low-resolution real terrain is synthetically enhanced to high-resolution. Such data is available for certain specific landing sites of interest. In addition to DEM's we must also mention data relevant to defining planetary geoid data. Geoid height is required to support atmospheric models. DSENDS supports both a DEM type representation of geoid data as well as those based upon a spherical harmonics expansion (cite Geoid paper).

A brute-force approach to pre-load a DEM of an entire region at high-resolution is not feasible (e.g.a swath of terrain 200 km by 100 km at 10 cm resolution is well over $2 * 10^{12}$ pixels). Instead a phased approach is required, where terrain patches are dynamically loaded into the simulation based upon the needs of the particular phase of the simulation. The phasing requires a number of components:

- dynamic generation of terrain patches at appropriate resolutions
- Fetching of generated terrain for use by the simulation

- Loading into special memory to facilitate real-time device response generation. Such loading and unloading must be transparent to the hard real-time process implementing the device simulation. In DSENDS this is achieved by means of utilizing shared-memory segments between the real-time process and a another thread responsible for terrain memory management.

- Look-ahead prediction of required terrain segments to allow

- Generation, fetching and loading requests to be queued and terrain to be available in a timely manner

DSENDS currently implements an Instrument Terrain architecture to support the needs identified above. The architecture is shown in Figure 2.

## Terrain Instrument Server

Terrain products are required within the EDL simulation to support a number of applications such instrument simulations, data monitoring modules, and 3-D visualization of the simulation. Examples of these are respectively, a terrain scanning Lidar instrument simulation, a monitor of the spacecraft height over the ground, and a 3-D view of the spacecraft approach to the landing site. The location, extent and spatial resolution of the terrain segments required to support these applications varies and is a function of the boresight, field-of-view, and the fidelity desired. For example, a Lidar with a steering mirror could require terrain anywhere within the field-of-regard provided by the mirror at a resolution that is a function of the instantaneous field-of-view (IFOV) of each pixel in the Lidar detector. A monitor of spacecraft altitude would require a small terrain segment located vertically below the spacecraft. A mouse-driven viewpoint generator for 3-D visualization would require terrain anywhere in the scene as the simulation user moves the view-point.

In all these cases terrain must be provided in a timely manner to the EDL simulator. This is especially important during real-time operations where instrument responses must be generated in synchronization with a real-time clock with no possibility of cycle-slips and consequent data loss. The option of having all of the terrain resident in memory for immediate access by the requesting EDL application is not feasible because of the sheer size of the data set required. For example, a 10km X 10 km site at 10 cm resolution would required storage of $10^10$ pixels ! Instead, a process of terrain generation (or enhancement in the case of synthetically augmented natural terrain) must be combined with terrain segment transport to the EDL simulator, and upload to the EDL simulator's memory. Each of processes identified has in-built delays that make timely access difficult. Fast generation of terrain using multiple fast processors, timely transport of data using fast network hardware and protocols, and real-time buffer and shared-memory segment management within the simulator, are all ingredients in achieving timely terrain access. Moreover, as the segments needed by the application change, successive terrain segments must be generated as needed, uploaded to the simulator, and placed into memory in a timely and seamless fashion.

The Instrument Terrain Server (ITS) in the EDL simulator provides these functions to the various applications in the simulator. It incorporates the following functional elements:

- The ITS has a number of real-time shared memory buffers which contain overlapping terrain segments. As the application requests terrain in the overlapping areas, buffers are switched in real-time to allow the application to access terrain in the new segment in a seamless fashion.

- The ITS has a predictive model of terrain usage (called an Oracle) that allows it to predict the extent , resolution and extent of terrain segments required by the application. These predictive models are usually based upon a nominal EDL scenario and the current location and velocity of the ground "footprint" of the instrument/viewer field-of-view.

- The ITS uses the predictions from the Oracle, knowledge of terrain generation times, data transport times, and buffer sizes to sequence the generation, transport and upload of appropriately overlapping segments of terrain into the EDL simulator. The ITS manages the use (and reuse) of the real-time buffers, the extent of overlap, and provides a level of cache management (e.g. keep adjacent terrain segments in memory in case they are needed) to relieve the simulator from frequent interactions with the terrain generation/transport process. Note that terrain generation can take many seconds, transport is usually a fraction of a second, and buffer management/swapping is done at simulation rates e.g 50 ms.

- In addition the ITS provides backup terrain (with lower resolution and larger spatial extent) in case the generation/transport process fails to achieve the times predicted by its model, or if the Oracle prediction of anticipated application terrain request turn out to be wrong (e.g. if an unexpected

spacecraft motion causes the Lidar steering mirror to hit a hard-stop and thereby cause the Lidar to suddenly be viewing a ground area far removed from the normal scenario).

A prototype version of the ITS is operational incorporating a Terrain software object, shared memory buffers that allow seamless ITS and application terrain buffer access, and a preliminary version of the Oracle (implement in Tcl). The ITS is being used to support a single instrument simulation application at this point (the Lidar simulation). Planned work will allow the ITS to support multiple EDL applications and refine the Oracle's predictive capabilities.

## AUTOMATION

Maintaining multiple, independent or interdependent states of various spacecraft components can be a daunting task, especially if the mission design is either in flux or not yet complete. An automated method which will simplify and partition the various control and data gathering functions involved with different components and mission phases is required.

The design of the DSENDS State Machine development was driven by the need to simplify dealing with different spacecraft stages as they separate, perform actions, or pass through different dynamic regimes. Since multiple, separate, spacecraft models can be active simultaneously, the state machine must maintain the state of each spacecraft or spacecraft component independently, and manage transitions between them.

Our state machine was designed based on providing a subset of common capabilities in existing commercial products providing similar functionality. The primary abilities desired were

- Automatic execution of user-functions during state execution and state transitions
- User-provided functions for testing of state transitions
- Multiple, simultaneous state capability
- Ease of definition and integration with DSENDS

In addition, the GraphViz graph visualization package was used to provide automatic display of the connectivity of the state machine, as well as to show the current active states in real time. Figure 12 shows an subset of the state machine driving the demonstration DSENDS simulation, showing names of the states and transition test functions, special functions like initialization (diamond) and termination of a state (rectangle), as well as the currently active states (green).

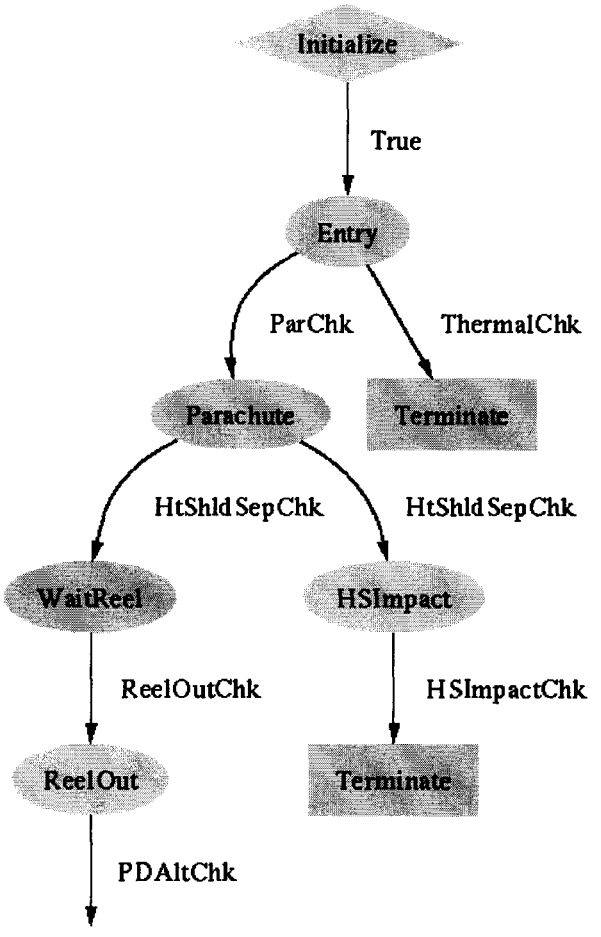


**Fig. 12 State Machine Visualization**

DSENDS batch processing capabilities are centered around performing automatic data collection for regression testing, trade-space analysis, and automated testing. The batch processor can parallelize execution of cases over a heterogeneous collection of interconnected processors, resulting in fast execution and data collection limited only by the number of processors available.

The Batch Manager does for data collection what the DSENDS State Machine does for the simulation: it streamlines and simplifies the process of running simulations for the purpose of data collection and analysis, and of generating answers to what-if questions. The batch manager provides a simple text-based interface that defines initial simulation states that are to be varied, like mass of spacecraft components, fule on board, entry angle, or spin rate. Then one defines the desired results to extract, which can range from fuel consumption, landing location, or landing error, to total horizontal delta-V. Finally, a list of available

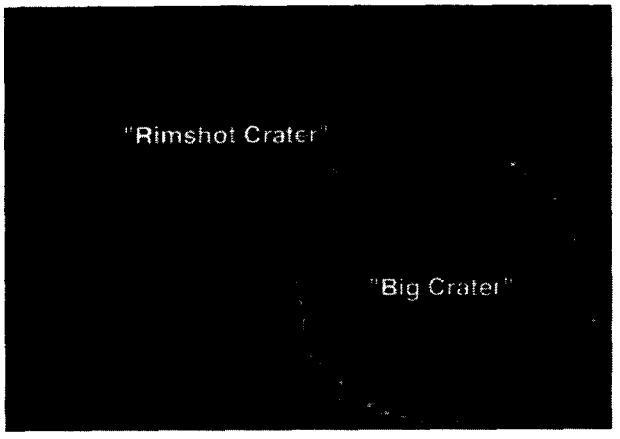


**Fig. 13 Sample Terrain**

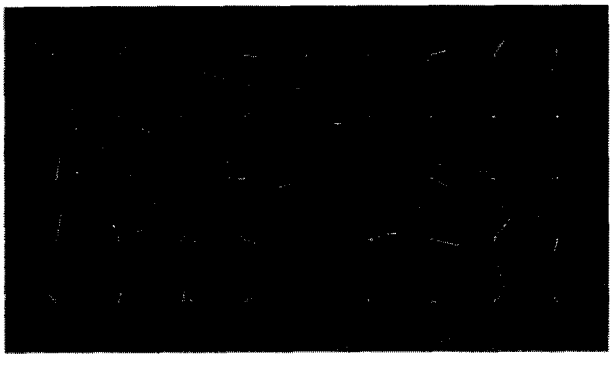

**Fig. 14 Batch Manager Output Analysis**

resources on which to run the simulations is provided. The input and output categories both use user-defined readable names to identify the cases to run and the output data.

The Batch Manager executes one simulation on each computer, then monitors the simulations and collects data as each completes. Once a resource is free, the next available case is run on it. This continues until all cases have been run and all data extracted, then the batch manager shuts down. Data is saved to an output file along the way, in case the process is interrupted.

An example of the output is show in Figure 14. In this sample, a 1 by 2 km cratered area was gridded with desired landing sites (green plus). A detailed spacecraft model was used that included a guided powered landing capability, with some hazard avoidance. A digital elevation map from the JPL MIPL laboratory was used that corresponded to the imagery. The output picture combines fuel consumption contours (white lines) and deflection maneuvers (cyan lines). The output data also indicated which sites were infeasible (red circles).

## Concluding Remarks

## References

[1]Gaskell, R. W., Collier, J. B., Hussman, L. E., and Chien, R. L., "Synthetic Environments for Simulated Missions," *IEEE Aerspace Conference Proceedings, Big Sky, Montana*, March 2001.

[2]Gaskell, R., Collier, J., Husman, L. E., and Chen, R. L., "Synthetic Terrain for Simulated Missions developed in a collaboration between Parallel Applications Technologies Group and Optical Navigation." *NASA Science Information Systems Newsletter*, , No. 60, July 2001.

[3]Lee, M., Weidner, R., and Lu, W., "Design-based Mission Operation," *IEEE Aerospace Conference, BigSky, Montana*, March 2000.